# Implementation of Beam Conditions Database for Intensity Frontier Experiments

**Igor Mandrichenko, Andrew Norman, Andrey Petrov, Vladimir Podstavkov**
FNAL, P.O. Box 500, Batavia, IL 60510, USA

**Abstract.** Neutrino physics research is an important part of the Fermilab National Accelerator Laboratory (FNAL) scientific program in the post Tevatron era. Neutrino experiments are taking advantage of high beam intensity delivered by the FNAL accelerator complex. These experiments share a common beam infrastructure and require detailed information about the operation of the beam to perform their measurements. We have designed and implemented a system to capture, store, and deliver this common beam data to all of the neutrino experiments in real-time. The solution that we designed and built is a robust, high reliability, high performance system that is capable of providing both real-time and historic beam conditions data to the experiments at different stages in their data acquisition and analysis chains. This system is currently being integrated into the online data collection, online monitoring and off-line data processing for each of the experiments. This paper will cover the design and implementation of this system, its interfaces.

## 1. Requirements

The Beam Conditions Database for Intensity Frontier Experiments (IFBeam DB) is required to store and make available information about FNAL accelerator beam conditions associated with every recorded event. It will be used by many Intensity Frontier experiments, as they research neutrino physics. Beam conditions data are critical for event reconstruction and data analysis, and need to be stored and made available throughout lifetime of the experiments, including the data analysis phase.

### *1.1. Data Volume*

The database receives and records data associated with several different beam extraction events. Peak event rate ranges from ~0.5 Hz to 15 Hz. The long term average event rate does not exceed 5 Hz. The amount of data per event is about 4000 double precision numbers or about 32 Kbytes. There will be multiple events recorded, so this number needs to be multiplied by factor of 5 or even 10. The expected data rate is up to 150 KBytes/sec.

About half of this data is used for online monitoring and therefore can be discarded after about a week. The rest has to be stored indefinitely. Long-term data flow will be up to 70 KB/second, or up to 2 TB/year.

The data will be accumulated over 5 to 10 years while the experiments are taking data. Therefore, the total size of the database at the end of its lifetime will be up to 20 TB of useful data, and it will occupy up to 40 TB of disk space per database instance, which includes database overhead and indexes.

### *1.2. Latencies*

Long-term data latency can be an hour or longer, because these data are used for data processing. Short-term data are used for online monitoring and should arrive in the database and made available much faster, within seconds.

### *1.3. Reliability*

Because the beams data are critical for the data processing and analysis, the database has very high reliability requirements. Data need to be collected and stored in reliable way, so that data can be delayed, but not lost due to such events as:

- Database downtime, planned or not
- Network downtimes
- Data collection component downtimes

In case of downtimes, it is acceptable to lose some short-term data, and to delay delivery of long-term data, but not loss of long-term data.

### *1.4. Flexibility*

It should be easy to change the configuration to start recording new data, associated with new events.

## 2. Design

IFBeam DB is designed as a modular system. Data collection is performed by redundant Data Collectors, stored in the Database and made available via Data Server.

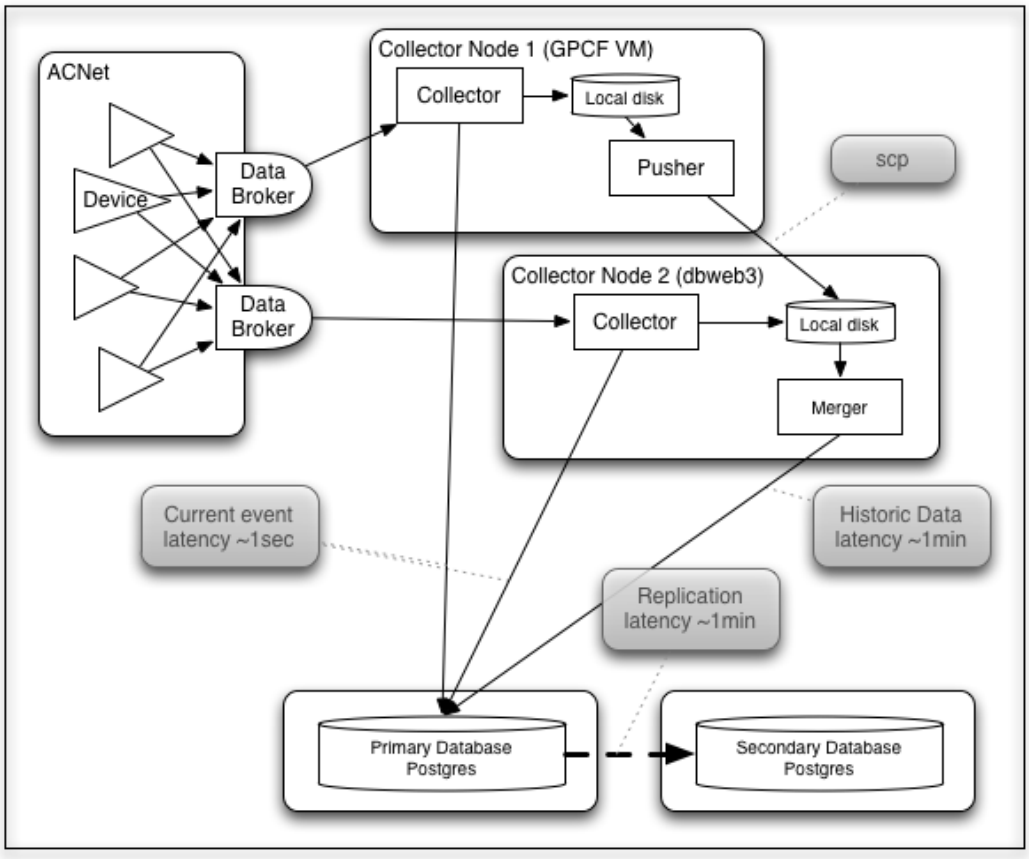


**Figure 1.** Data Collector Design

*2.1. Data Collector*

The IFBeam DB Data Collector's function is to receive data from the accelerator control system (ACNet) buffer it locally if necessary and deliver it to the database (see Fig. 1). The system is designed to run multiple, completely redundant collectors. Each collector receives data in real time from ACNet and then delivers them in 3 different ways:

- Slow – data is stored in local flat file. Periodically, about once per minute the file is closed and sent by the Pusher to the merger node. The Merger eliminates redundancy in data and stores it in the database, in long-term and medium-term tables.
- Fast – data is sent immediately to the database where it is stored in a special table, which holds only one last event.
- A UDP message with the event timestamp only is sent to the web server for monitoring purposes.

Each collector has sufficient disk space to buffer a week or more worth of data in case there is a network interruption, or the merger node is down, so the data can be buffered for later delivery. If the database is unavailable and fast delivery mechanism is not working, the latest event is discarded. If the network is down, UDP messages with event timestamps are discarded by the TCP/IP stack.

Data Collector is implemented in Java because it uses ACNet Java client API. The rest of Collector is written in Python.

*2.2. The Database*

We use PostgreSQL [1] 9.1 as the database engine. The IFBeam Database runs on 2 identical redundant servers (see Fig. 2). Each server has 8 1TB disks; 6 out of 8 disks are combined into RAID-10 (striping and mirroring) 3TB logical volume. The 2 remaining disks are not RAID'ed. We use RAID'ed space to store long-term data and one of 1 TB disks to store short-term data. Storing long-term data on mirrored disk protects the data from a disk failure.

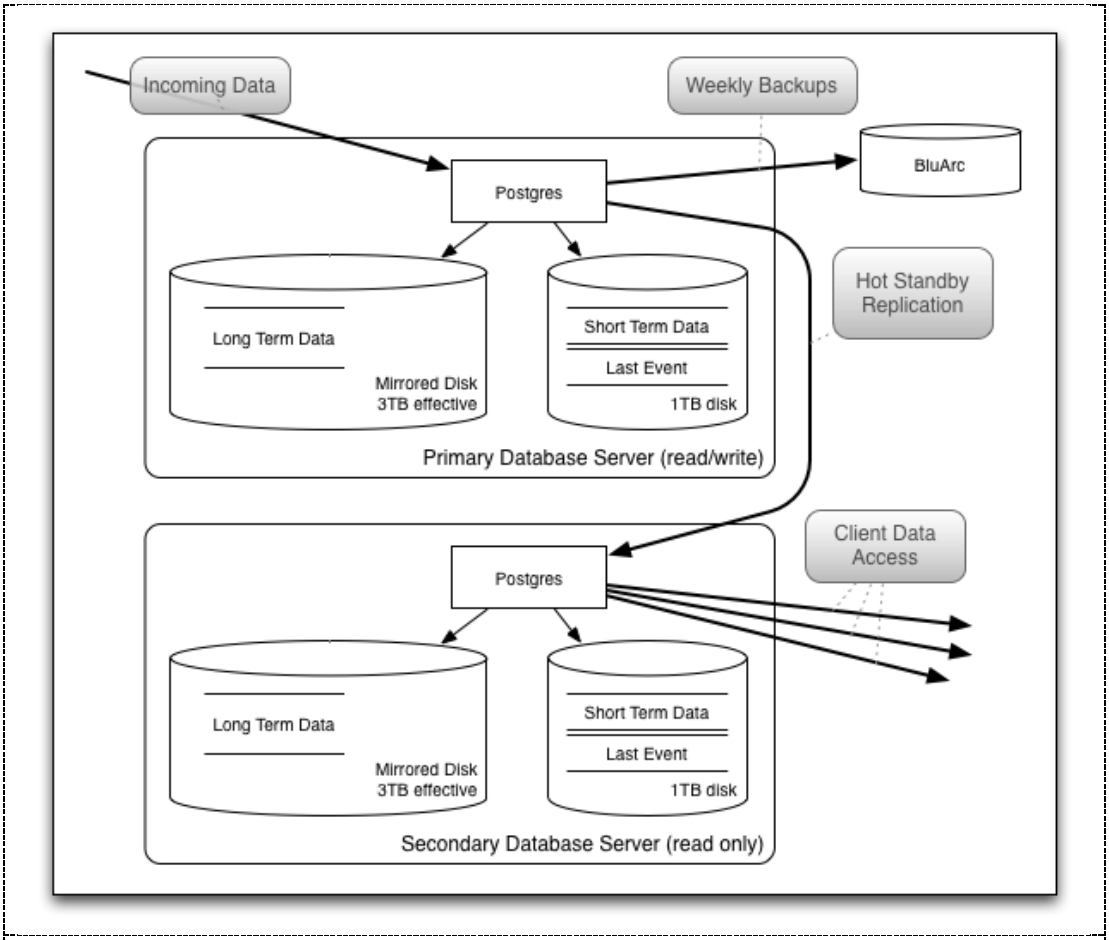


**Figure 2.** Database design and data replication

In addition to data replication within the server, we use PostgreSQL hot standby replication [2] between the two severs. This way the database is protected from individual server failure. In case of permanent failure of a server, we will reconfigure the system to run using single server until the other server is replaced.

Data are accessed only from the secondary server, so that the primary server is dedicated to receiving data from the Data Collector.

With disk mirroring and the data replication between 2 servers, our long-term data are replicated 4 times and short-term data 2 times. In addition, we set up a weekly data backup to external disk storage. This backup will be used in case of a catastrophic loss of both database servers. We use standard PostgreSQL backup/recovery procedure, and copy data using rsync, which copies only the portion of the database storage which has changed between the backups.

*2.3. Data Access*

Although direct access to the database is technically possible, users are strongly discouraged from using the database in this way because of the scalability concerns. Instead, the standard method of accessing the data is via the web-based Data Server. The Data Server (see Fig, 3) runs on 2 redundant web server computers as a WSGI [3] application under Apache httpd [4]. It is implemented in Python.

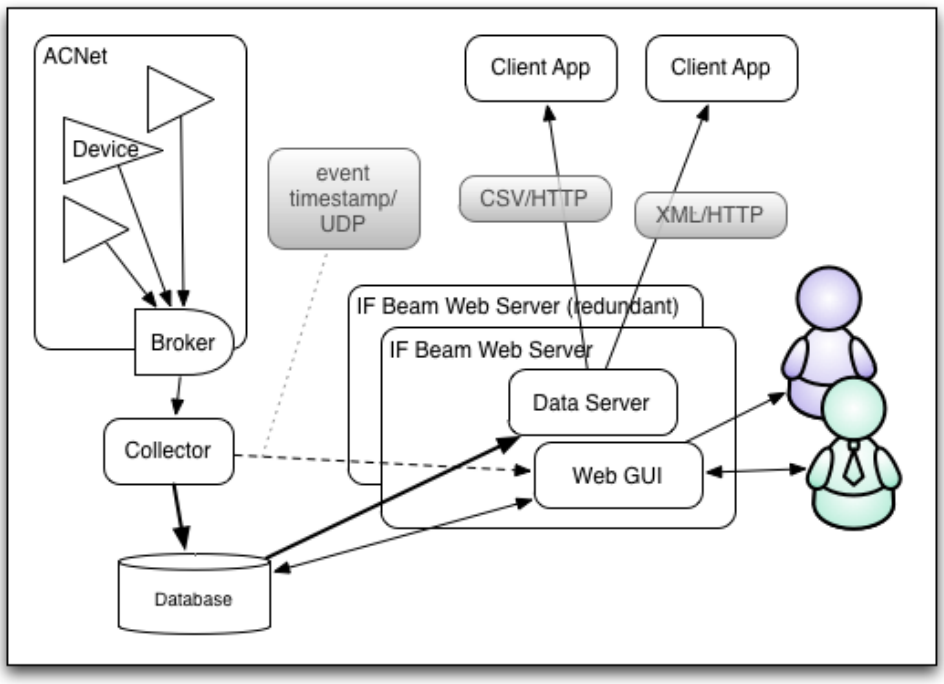


**Figure 3.** Data access

The Data Server exports data in response to HTTP requests in CSV, XML or JSON formats. In addition to the long-term and short-term data, the Data Server provides real time access to the event timestamp information received via UDP message from the Collector.

Using redundant web servers (see Fig. 4), we achieve high availability of this service. We use the HTTP Redirector to multiplex between individual web servers. The Redirector receives the request from the application, chooses one of available data servers by sending them simple “probe” request at configured URL and redirects the client to the server it found. It uses a simple round-robin method to distribute the load between the servers.

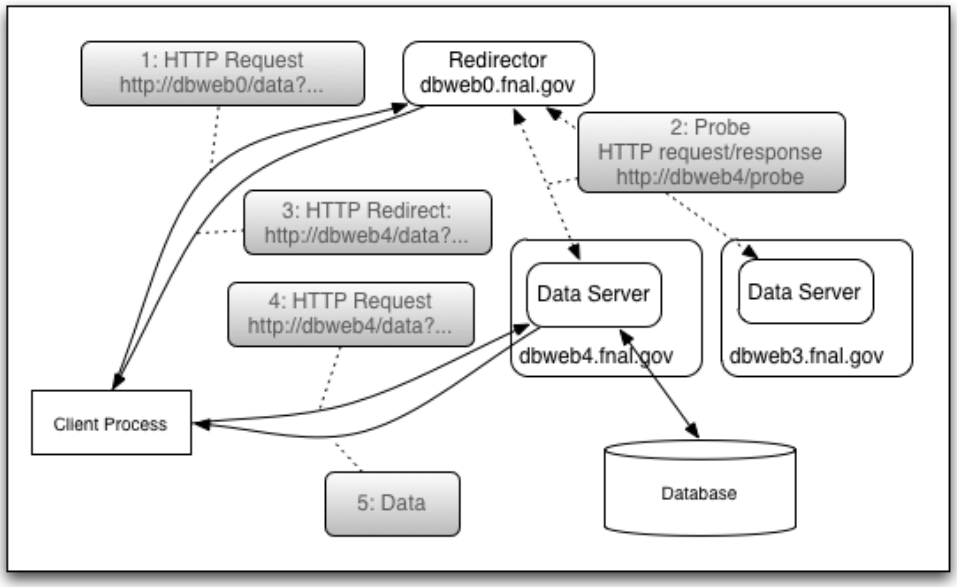


**Figure 4.** Redundant web server infrastructure

*2.4. Monitoring*

We use web-based GUI to monitor health of he system. Our dashboard (See Fig. 5) shows real time data received from the database through the secondary database server and using the same web Data Sever interface as the client applications. The dashboard shows event frequency charts as well as time graphs of several most important beam condition variables. This allows us to detect malfunctions occurring at any point in our data collection and storage pipeline.


**Figure 5.** IFBeam Database monitoring dashboard

## 3. Project Status and Conclusion

The IFBeam DB project started collecting production data since September, 2011. By June, 2012, we have collected 57 million events, 411 million data measurements. The current database size is about 300 GB.

We have built a scalable, high performance system to collect, store and make available beams conditions data for multiple neutrino experiments running at FNAL. Due to redundancy, the system is very robust and reliable and is designed to continue functioning and meeting its requirements in wide range of cases of failures of individual components, is easy to maintain.

## 4. References

[1] http://www.postgresql.org
[2] http://www.postgresql.org/docs/9.0/static/hot-standby.html
[3] http://code.google.com/p/modwsgi/
[4] http://httpd.apache.org/docs/2.0/programs/httpd.html

**Acknowledgments**
Authors wish to acknowledge assistance and encouragement from colleagues and valuable input from members of neutrino experiments.